AFHRL-TP-90-66 DTIC FILE COPY

(2)

AIR FORCE

AD-A226 729

HUMAN RESOURCES

AUTOMATED DESIGN OF DISPLAYS FOR TECHNICAL DATA

Stephen Westfold
Cordell Green
David Zimmerman

Kestrel Institute
3260 Hillview Avenue
Palo Alto, California 94304

LOGISTICS AND HUMAN FACTORS DIVISION
Wright-Patterson Air Force Base, Ohio 45433-6503

September 1990

Interim Technical Paper for Period June 1989 - May 1990

Approved for public release; distribution is unlimited.

LABORATORY

AIR FORCE SYSTEMS COMMAND
BROOKS AIR FORCE BASE, TEXAS 78235-5601

90 09 21 011

NOTICE

DAVID R. GUNNING
Task Monitor

BERTRAM W. CREAM, Technical Director
Logistics and Human Factors Division

JAMES C. CLARK, Colonel, USAF
Chief, Logistics and Human Factors Division

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE
September 1990

3. REPORT TYPE AND DATES COVERED
Interim Technical Paper - June 1989 - May 1990

4. TITLE AND SUBTITLE
Automated Design of Displays for Technical Data

5. FUNDING NUMBERS

C	-	F33615-88-C-0004
PE	-	63106F
PR	-	2950
TA	-	00
WU	-	21

6. AUTHOR(S)
Stephen Westfold
Cordell Green
David Zimmerman

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Kestrel Institute
3260 Hillview Avenue
Palo Alto, California 94304

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)
Logistics and Human Factors Division
Air Force Human Resources Laboratory
Wright-Patterson Air Force Base, Ohio 45433-6503

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFHRL-TP-90-66

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This paper reports on a theory of a graphical display that automatically designs tables and diagrams that represent relations occurring in technical data. The theory is based upon a small number of principles such as using visual adjacency and visual connectedness to reflect semantic relations among the data. The theory uses techniques adapted from automated programming for display reformulation and display optimization. A prototype display synthesis system that implements the theory as a set of transformation rules has been written. The system takes a relational specification as input and produces a display as output. The theory is illustrated in this paper by showing an annotated trace of the prototype generating a large number of tables and diagrams from relational technical data. The examples show the generation of displays from aircraft maintenance technical data that is free of all formatting and display instructions.

[illegible]

14. SUBJECT TERMS
automated generation of formats
automated technical data
computer generated maintenance aids

15. NUMBER OF PAGES
46

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

AFHRL Technical Paper 90-66 September 1990

AUTOMATED DESIGN OF DISPLAYS FOR TECHNICAL DATA

Stephen Westfold
Cordell Green
David Zimmerman

Kestrel Institute
3260 Hillview Avenue
Palo Alto, California 94304

LOGISTICS AND HUMAN FACTORS DIVISION
Wright-Patterson Air Force Base, Ohio 45433-6503

Reviewed by

Robert C. Johnson
Chief, Combat Logistics Branch

Submitted for publication by

Bertram W. Cream, Technical Director
Logistics and Human Factors Division

This publication is primarily a working paper. It is published solely to document work performed.

SUMMARY

This paper describes a prototype computer software system automating the design of computer displays. The information displayed includes tables (e.g., as parts tables or diagnostic tables) and functional diagrams.

The concept of operations for the Integrated Maintenance Information System (IMIS) requires that all technical data be stored in a database independent of screen formatting information. For example, the electronic form includes no information about where a box of the diagram is to be placed, or even that some information is to be displayed as a diagram instead of a table. Hence, this *format-neutral* storage representation then allows software to automatically reformat the data for new computer screen sizes, new portables, new color capabilities, or new methods including animation or head-mounted displays. This separation of information content from formatting also permits better analysis to check for consistency or to merge information from multiple sources. When there are changes, minimal human reengineering effort is required to update the information and the displays, resulting in reduced costs and higher quality.

The automatic design and layout methods presented here represent a new capability. Previous layout techniques exist but have been limited to a few preselected screen structures whose rendering does not adapt to new data. In addition, these previous methods do not adapt to algorithmic techniques integrated with human factor constraints to select high-quality, dependable displays.

Our approach may be termed *rule-based*, since each table or diagram is automatically generated from a set of rules dictating what constitutes an acceptable rendering on the screen. The rule-based approach makes it easy to incorporate human factors rules for guiding the selection of appropriate displays.

The capability to build a computer system to carry out these logic-based designs has been enabled by advances in theory and in technology including very-high-level languages, more advanced compilers, and representation transformation tools.

The main principle used by the rule-based system is that items associated in the database, such as a symptom-fault pair, must be given a visual association on the screen, such as occurring in the same row of a table. Elaborations of this and related principles are presented and illustrated in this paper.

Our rule-based prototype has been demonstrated with maintenance technical data provided by the Air Force Human Resources Laboratory (AFHRL). This paper presents a series of tables and functional diagrams automatically designed by the prototype computer software system. The tables contain fault, symptom, and test data, and the functional diagrams are for an aircraft subsystem. Paper (not electronic) versions are also produced automatically (the tables and diagrams in this paper were produced this way).

PREFACE

The work described in this technical paper was performed by the Kestrel Institute under subcontract from Systems Exploration, Inc. (SEI). The work was accomplished for the Air Force Human Resources Laboratory under contract F33615-88-C-0004, task number 10. The work is in support of AFHRL's Project 2950, Integrated Maintenance Information System. Mr. David R. Gunning monitored the contract for AFHRL.

This paper is concerned with the automated display of neutral maintenance data. The display technology described in this paper establishes the feasibility of this approach for the IMIS and provides an initial prototype for evaluation.

The prototype is written in the REFINE™ language, running on Sun 4 workstations. The drawing routines and the user interface were implemented within the X-11 window system running under INTERVISTA™.

The original basis of this work came from an idea suggested by Cordell Green and elaborated into a suggestive set of rules by David Gunning for a class project at Stanford University. We would also like to acknowledge the guidance of David Gunning in (a) providing the initial theoretical foundation for our logic-based approach while he was a graduate student at Stanford, and (b) showing how our research could be applied to the IMIS technical data design problem.

The transformational theory for tables developed and presented herein is from work by Stephen Westfold, Cordell Green and Janet Coursey. Stephen Westfold and Janet Coursey implemented the table generator. David Zimmerman and Stephen Westfold designed and implemented the diagram generator. David Zimmerman and Li-mei Gilham implemented the user interface. Westfold and Zimmerman put together all the pieces to build the integrated prototype system.

We would like to acknowledge Mark Miller of SEI in helping us through the myriad of technical and administrative obstacles we encountered doing this work. We acknowledge the valuable time spent by Li-Mei Gilham developing our user interface. We acknowledge Jorge Phillips' efforts coding the box-rendering routines, Marilyn Daum's design and implementation of a menu system, and Dick King's help with SGML and authoring. We also acknowledge assistance from Lt. Janet Murphy of AFHRL, and Doug Andrews and Ross Rannels of Systems Research Laboratories.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

Recent advances in technology for greater automation in software include knowledge-based methods for software creation and logical specification of software using very-high-level specifications (Smith, 1988), (Kotik, Rockmore, & Smith 1986). This advanced technology can be applied to the automated generation of graphical user interface software. The key observation is display synthesis is analogous to software synthesis.

We illustrate herein a theory for the creation of tables and diagrams. We plan to achieve for automated display synthesis the large increase in productivity and maintainability automated programming brings to software production and maintenance. Display or presentation specifications are software, too and are subject to modification and in need of maintenance. Users want fast and inexpensive turnaround to test requirements before making a larger investment.

Such automated design of graphical representations can (a) increase productivity, (b) allow the rapid exploration of a large number of alternative display designs, and (c) be used to produce standard display formats from unformatted data. It allows for the generation of displays for unanticipated cases (i.e., a rule-based design system effectively determines what kind of displays can be effective in communicating new information).

The significance of this technology for the Integrated Maintenance Information System (IMIS) project is severalfold. First, the IMIS will apply the concept of storing *format-neutral* data in the repository of technical data. The data are neutral with respect to display and formatting. Thus, as new display consoles, new maintenance workstations, new screen look-and-feel standards, and new printing standards emerge, the data can automatically be formatted to comply with these new constraints. By driving all the display and print systems off the same neutral data, consistency is established across the various media. As new techniques such as color, animation or virtual reality displays are introduced, little or no change to the technical data will be required. The availability of a low-cost display design capability allows the technical data to be freed from formatting information so inference systems can analyze the data for consistency, quality, optimization, upgrades, and so on. Also, technical data from multiple vendors can be merged into combined maintenance presentations using data structure and procedure reformulation techniques.

This work differs from the conventional notion of a "screen generator." The generative theory adapts to new types of data or display criteria because it uses the structure and contents of the data to guide the creation of the display. We illustrate these capabilities in detail in this paper.

Two main categories of choice are provided by the synthesizer: (a) data structure reformulation, and (b) variation of display styles. The data reformulation steps reorganize the information directly so the system can synthesize a greater variety of displays than is possible by a system that fits the data within a predefined display format.

There are subsystems for generating *tables* and *diagrams*. These two subsystems share the same graphics bounding-box model and implementation as well as many of the same design principles.

Organization of the Report

Section II describes our technical approach. Section III describes the table generator in more detail and presents an annotated trace as the prototype designs a series of tables. Section IV describes the diagram synthesis prototype. Again, several examples of generated displays help illustrate the process. Section V suggests several future directions and contrasts this research with some related work in the area. The last section offers our conclusions regarding this first phase of work.

II. TECHNICAL APPROACH

Generator/Selector Factoring of Theory

We are interested in automated design assistance for the generation of graphical user-interface software. The research strategy chosen is to divide the problem into a *generator* theory and a *selector* theory.

A *generative theory* takes as input some information to be displayed graphically and generates as output *all plausible* graphical ways to display that information. Such a generative theory forms the basis for the automatic synthesis of graphical displays. The degree to which the theory can generate all plausible displays determines the theory's flexibility and utility.

A *selector theory* takes as input a set of displays and places as output the displays in order of preference (or just names one that is satisfactory). The selector theory consists of a set of criteria by which the displays are filtered so preferred displays are separated from the many possible displays generated. In the implementation, the selector prunes the space of possible displays proposed by the generator.

We are following the research strategy of first pursuing the development of the generative theory and later combining it with a selector theory. Accordingly, in this research effort we have taken the important first step and developed the *generative theory* for the automated synthesis of tables and diagrams.

Human factors determine the selector theory. For example, the applicable human factor analysis may call for limits on the ratio of line width to character size, or on the number of functional blocks shown in a limited space. Such criteria would be stated in the selector theory.

This division of the theory into a generator and a selector is used for theory development and explanatory purposes. Thus, the concept is simply that the generator produces the space of presentations and the selector prunes this space. However, for efficiency, the implementation can be made more efficient by incorporating the selector rules into the generator (enabling rapid pruning of the design tree).

To limit the scope of this initial investigation into the theory, we have restricted the type of graphical displays generated to forms of tables and diagrams (graphs). It does not include animation, color, or solid modeling, although we see no intrinsic limitations on extending the methodology into these areas.

Analogy to Program Synthesis

Graphical screen synthesis is quite analogous to conventional program synthesis. In particular the two notions of *data structure reformulation* and *program optimization* are directly applicable to graphics synthesis. For example, the common program optimization that consists of pulling constants out of a loop has a direct analogue in graphics, such as removing constant expressions from visual representations of sets or sequences and only presenting the constant expression once (e.g., as a heading for a column of elements).

In addition to these particular principles that carry over directly, the larger methodology of program synthesis also fits graphics synthesis. One example is the use of our technology of transformations to transform internal information structures into external screen entities. Another example, analogous to the display generator and selector discussed above and found in conventional program synthesis, is an efficiency estimator (selector) needed to guide the heuristic search for an efficient implementation since all possible implementations (produced by the generator) cannot be searched. Graphics synthesis also needs an efficiency estimator—one that estimates how well the ultimate execution engine of the graphics, namely the human user, will be able to "execute" or use the graphics structure to solve the problem.

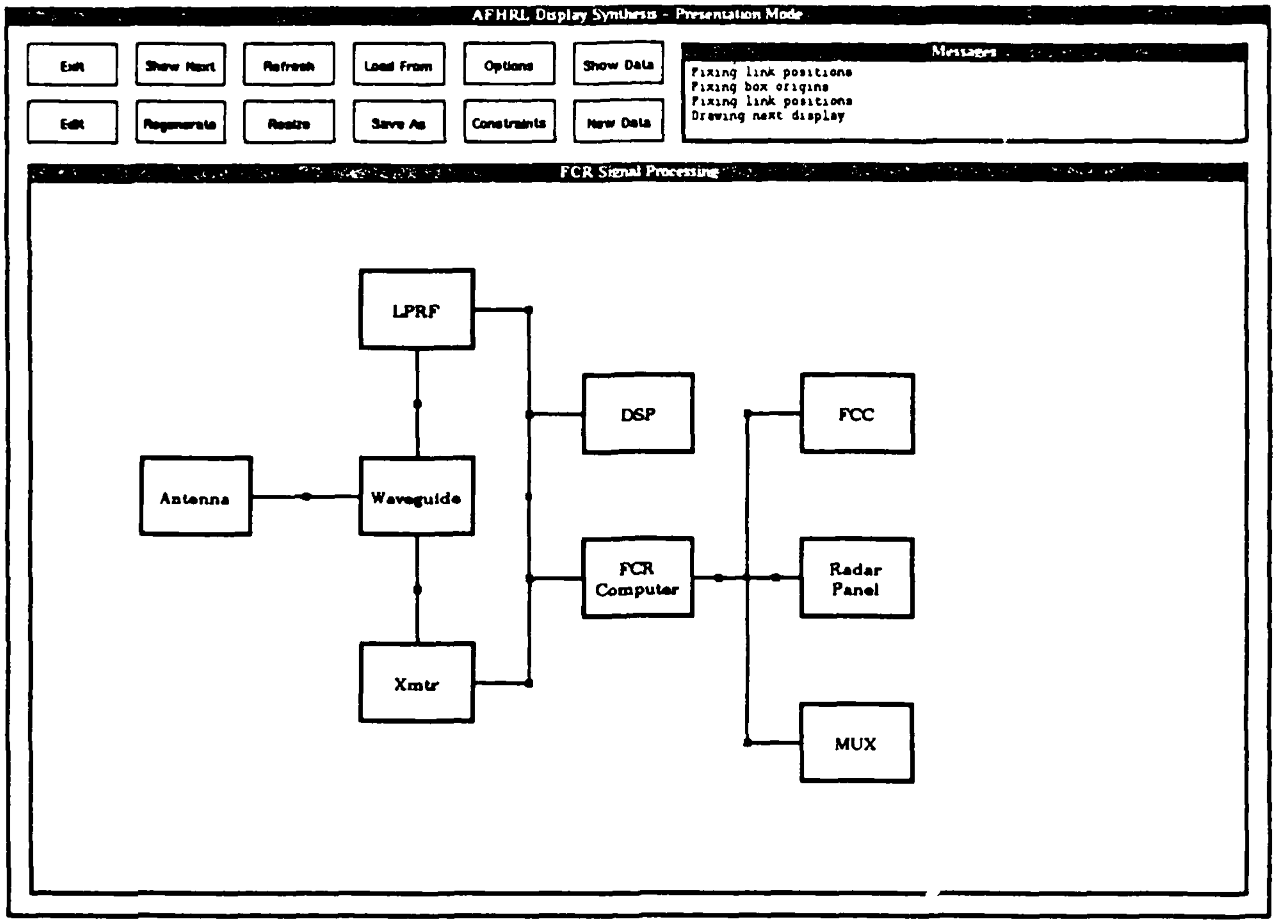
Progress

We have developed the rule base necessary to synthesize programs that design displays of relations as diagrams and tables. We have been able to test this theory by implementing a simple prototype graphics generator proving the concept.

The problem has proved to be tractable with our methodology and we are optimistic about the future possibilities. The theory has fulfilled our hopes in several ways: (a) the theory did indeed turn out to be simple (after we had worked it out), (b) we were able to build upon our transformational foundation and experience in automatic programming to gain considerable leverage, (c) the implementation of a prototype was straightforward, consisting of expressing the graphics theory as a set of compilable transformation rules, and (d) the theory can be immediately extended in several directions, as discussed in the conclusion of this paper.

We have applied our prototype display generator to several forms of input including sets, mappings and relations. It produced many different types of tables and diagrams. The example in Section III shows a series of tables automatically generated from a binary relation. The examples in Section IV show diagram layouts automatically generated from just the connectivity information.

Since paper "displays" are also a target of this work, we have implemented a preliminary set of rules for producing printable representations (as TEX files) of generated displays. We have also constructed a menu-based interface to the generator, which allows the user to select various options, enter new input, and cycle through a generation sequence. A screen dump of this interface is shown in Figure 1. All of the diagrams and tables reproduced in this paper were generated automatically by the system and used as is.



```
Fixing link positions
Fixing box origins
Fixing link positions
Drawing next display
```

Figure 1. Menu-Based Interface to Display Generator

Guiding Principles

This theory was inspired by the observation that it seems possible to generate displays using only a few simple principles. One idea is to show that two items are associated to show a simple visual association. That is, internal commonality can be communicated by visual commonality. For example, if two objects in a database share a common property, such as occurring in the same field of a relation, then that association can be shown on the screen by having the two objects share a common screen property such as horizontal or vertical position.

One step further than this simple pairwise association is the notion of ordering. An ordering in the data can be communicated to the external world by employing some sort of visual ordering, such as increasing the y position, intensity, or angle.

Two basic principles guide the synthesis of the display: (a) visual encoding of the semantic role of the data structure's parts, and (b) optimization. The contents of the data belong to the same structure because they share some common property or have a certain relationship with each other. The visual display should convey this information.

The display should strike a balance between useful redundancy and unnecessary clutter. Optimization helps remove undesirable redundant parts of the presentation, and make the best use of the selected visual display properties.

1. **Semantic and visual similarity.** A shared property of the items being displayed is shown as some shared visual property on the screen—vertical alignment, horizontal alignment, being joined by arcs, or table position. Prettyprinting is an example in which the items at the same depth in the expression tree share the visual property of the same level of indentation.

The semantic properties of the data structures in our examples include membership in a set or sequence, and fields of the same tuple. A subclass of sameness is *association* between items, as defined by a mapping or a relation. It is reflected as a visual association: (a) horizontal or vertical juxtaposition, (b) "labels" or, (c) line segments joining the items.

2. **Optimization.** When displaying complex data, or data with labels, parts of the relation or the labels may appear multiple times because they are associated with each datum. A common subexpression removal type optimization can eliminate some of the repeated patterns. This optimization is analogous to standard compiler optimization techniques, (such as finite differencing, pulling constant calculations out of loops, loop combining, etc.,) that identify and remove repeated computations from programs.

3. **Well-ordering.** Often the information to be displayed is ordered (alphabetically, numerically, by region, and so on). Even if the ordering is not explicitly present in the data, it is sometimes useful to choose an ordering principle to help structure the display. The intended use of the data will govern the ordering for example, telephone directories ordered alphabetically by surname or by street address. A radar display uses a radial ordering to sweep out the field of view.

Our goal is to develop a robust generator. This effort requires a very fine-grained theory: the ability to generate all the plausible variations of graphic displays from all plausible data. The fine granularity of the reformulation steps and the details within each display style is what guarantees adequate coverage of the space of possible graphic representations. Restated, we want the grain size of the display decisions (such as whether to center or justify text, font choice, and so on) to be fine enough to generate a wide spectrum of displays. Then some of the displays are likely to be particularly effective for communicating a given set of information. We can characterize such a theory as robust, not brittle. In contrast, a generator that selects displays from several large, precanned display types cannot be gracefully extended. By allowing subtle variations to be generated, we have the flexibility to handle new data and new types of display media.

Simple Examples

We now look at two displays for small data structures to illustrate the application of the association and optimization principles.

An element in a complex data structure plays one or more roles in relation to the whole. For example, the elements of a set are in the membership role with respect to the whole. Corresponding domain and range elements from a map are in one role of associating one with the other, and another role of being one entry from a larger map. In displaying a data structure, we choose some of these roles to manifest visually. Some visual trait will be selected to indicate each chosen role. The possible choices for the visual sameness are same-row, same-column, same-font, arc-plot (tables with line segments joining related entries), and offset-plot (tables indexed by row and column position).

Tuple

For a pair $\langle a, b \rangle$, the property shared by a and b is membership in the same tuple; that property could be visually represented as either vertical or horizontal alignment (same-column or same-row):

```
                        a
        a b     or
                        b
```

Sequence

The increasing order of the elements in a sequence can be shown by *incrementing* some visual trait (e.g., vertical or horizontal position, intensity, size, and so on) as each element is displayed. That the elements belong to the same sequence can be shown by holding a visual trait *constant* for each element.

To show a sequence, the "elements" relation is shown by holding constant one trait; the "ordering" relation is shown by varying another trait. For example, the vertical position can be kept constant while the horizontal varies, shown first below, or the role of the horizontal and vertical reversed, yielding the second display below.

For the sequence $[1,2,3]$:

```
                        1
        1 2 3    or     2
                        3
```

Two other association possibilities for maps are shown in the extended example: (a) using line segments to link related items, and (b) using a plot chart.

Visual Representations

In this section we consider in more detail available visual representations and how they can be used.

To display a set of relationships adequately, we must represent every such relationship visually and not have any other relationship implied by the display.

The visual representations considered in this paper are the following:

1. A horizontal or vertical positional association can represent a tuple, a sequence or a set.

2. A connecting line can represent a pair.
3. A plot chart can represent a map $A \times B \mapsto C$.
4. A box may represent any complex object.
5. A visual attribute (e.g., color, shape, or font) may represent a pair.

Note that we consider a pair to be a special case of a tuple (a 2-tuple).

It is necessary to represent the objects themselves as well as the relationships between objects. In this paper we always represent objects by text strings (possibly boxed).

A good display should minimize the number of times any individual object is represented. However, it is often desirable to represent a relationship redundantly. For example, in diagrams where relationships are always represented by lines, it is desirable to represent relationships positionally as well; and, in complex tables where sets and tuples are represented positionally, representing them additionally by boxing can prevent an accidental positional alignment from being misinterpreted as an additional relationship.

The Relational Language

The language to express the relational information[1] has the following constructs:

- $\{x, y, \ldots\}$: the set consisting of $x, y, \ldots$
- $\langle x, y, \ldots \rangle$: the tuple consisting of $x, y, \ldots$
- $\lfloor x, y, \ldots \rfloor$: the *collection* consisting of $x, y, \ldots$
- $\{f(x) | p(x)\}$: the set consisting of $f(x)$ for every x such that $p(x)$.
- $\{|f(x) \mapsto g(x) | p(x)|\}$: the (partial) map that maps $f(x)$ to $g(x)$ for every x such that $p(x)$.
- "a-string": the string *a-string*.

A *collection* is different from a set in that all the elements need to be displayed but they do not have to be visually associated. For example, the elements of a set must be displayed consistently, such as in a single row or column, whereas the elements of a collection may appear at any relative position on a page or on different pages.

The following is an example specification of relational information:

[1]The language does not include relations as a primitive type; instead, relations are modeled as sets of tuples.

$\lfloor$⟨"Symptom", $\{x \mid \langle x, y\rangle$ in *Diagnosis-Relation*$\}$⟩,
⟨"Fault", $\{y \mid \langle x, y\rangle$ in *Diagnosis-Relation*$\}$⟩,
⟨"Bus Problems", $\{\langle x, y\rangle \mid \langle x, y\rangle$ in *Diagnosis-Relation*$\}$⟩$\rfloor$

Considered as a specification of information to be visually displayed, this structure can be read as follows. Display three structures: first, a tuple with the first element (the label) *Symptom*, and with the second element being the set of all x, whenever $\langle x, y\rangle$ is in the set Diagnosis-Relation; second, a tuple with first element *Fault*, and with second element being the set of all y, whenever $\langle x, y\rangle$ is in the set Diagnosis-Relation; and third, a tuple with first element *Bus Problems*, and with second element being the set of all tuples $\langle x, y\rangle$, whenever $\langle x, y\rangle$ is in the set Diagnosis-Relation.

The set comprehension in the third tuple is just the set Diagnosis-Relation. This expanded form is used to explicate the structure of the relation so it may be transformed by simple structural transformation rules.

The value of the Diagnosis-Relation that we will use in our examples is:

Diagnosis-Relation =
{⟨"Bad data received", "defective board"⟩,
⟨"Bad data received", "bad sensor"⟩,
⟨"Transmission fails", "card in wrong slot"⟩,
⟨"Transmission fails", "defective board"⟩,
⟨"Bus error signal", "card in wrong slot"⟩}

Note that this specification of *Diagnosis-Relation* is *concrete* in that it does not have any variables in it, whereas the prior example is *abstract* in that it depends on the variable *Diagnosis-Relation*. We will use these specifications in our examples in the next section.

Tables and Diagrams

The system we have implemented takes specifications of relational information in the language presented in the previous section and displays them so as to reveal their structure. Our system has two subsystems: (a) one for generating tables, and (b) one for generating diagrams. The table generator works with abstract specifications, and thus produces display structures are for the most part independent of the actual values of the variables (i.e. they are data-independent). The diagram generator works with concrete specifications and thus produces data-dependent displays. In particular, the diagram generator works with binary relations where all the structure is in the data.

The table generator has an interpreter that takes an abstract specification and produces a display with the same structure as the evaluated specification. Thus, any repetition of objects in the evaluated specification will cause repetition in the display. If the abstract specification is transformed to remove a redundancy, then the display derived from the transformed specification will have that redundancy removed.

The diagram generator displays relations using links to connect the related objects. Each object and link is only displayed once so the generator is not concerned with removing redundancy. Instead it is concerned with layout to allow simple, short links as much as possible, and to allow redundant use of positional adjacency to strengthen the visual association for the viewer. Indeed, if two associated objects are positionally adjacent, then a simple link may be drawn between them, so these two criteria are complementary.

Thus, the table generator is primarily working on redundancy removal and has simple layout strategies, whereas the diagram generator is primarily working on layout and need not be concerned with redundancy removal.

III. TABLE SYNTHESIS

The table synthesizer uses three kinds of rules: (a) representation rules, (b) redundancy removal rules, and (c) conditioning rules applied specifically to enable application of rules of the first two kinds.

Representation Rules

In addition to the three primary kinds of visual association described in the following three subsections, boxing is used around sets and tuples except at the top level. Boxes containing boxes are given thicker lines than the boxes within them. This representation redundancy is sufficient to nullify accidental positional alignments.

Positional Association

A tuple or a set may be represented by horizontal or vertical positional association. If the parent of a tuple or set is represented by horizontal or vertical association then the opposite is chosen. This simple rule is too restrictive in general but is sufficient for the kinds of examples considered in this paper. The top-level choice is unconstrained so the generator will generate two identical displays with x and y positions reversed.

For example, if the top-level of the specification

$\{\langle x, y\rangle \mid \langle x, y\rangle$ in *Diagnosis-Relation*$\}$

is represented by vertical association then the tuple will be represented by horizontal association, giving the first display in Figure 2. The second display results from the reverse choice.

We will show henceforth only one of these symmetrical pairs of displays in each case. However, this variety is important. Sometimes one orientation will not fit on a page but its symmetrical version will. Sometimes it is more convenient for a user to locate information by searching down a column rather than across a row.

Connecting Line Association

A set of pairs may be represented visually by displaying the set of first elements of the representation, the set of second elements of the pairs, and a line for each pair connecting the first element

Bus error signal	card in wrong slot
Transmission fails	defective board
Transmission fails	card in wrong slot
Bad data received	bad sensor
Bad data received	defective board

Bus error signal	Transmission fails	Transmission fails	Bad data received	Bad data received
card in wrong slot	defective board	card in wrong slot	bad sensor	defective board

Figure 2. Symmetrical Displays of Relation

of the pair with the second element. This representation is the basis of the diagram generator. We include a special case in the table generator that does not require data-specific layout (although it may benefit from it): display the first elements vertically and next to them display the second elements vertically with space in between them where the connecting lines are drawn. It is useful if the set of first elements is disjoint from the set of second elements and there are not too many line crossings.

The rule performs the transformation

$$\{\langle x, y\rangle \mid p(x, y)\}$$
$$\longrightarrow$$
$$\lfloor \{x \mid p(x, y)\}, \{y \mid p(x, y)\}, \{\langle x, y\rangle \mid p(x, y)\} \rfloor$$

along with annotations to specify the display methods of the components.

For example, the specification

$$\{\langle x, y\rangle \mid \langle x, y\rangle \text{ in } \textit{Diagnosis-Relation}\}$$

is transformed to

$$\begin{aligned}&\lfloor \{x \mid \langle x, y\rangle \text{ in } \textit{Diagnosis-Relation}\},\\ &\ \{y \mid \langle x, y\rangle \text{ in } \textit{Diagnosis-Relation}\},\\ &\ \{\langle x, y\rangle \mid \langle x, y\rangle \text{ in } \textit{Diagnosis-Relation}\}\rfloor\end{aligned}$$

resulting in the display of Figure 3. This figure illustrates an exception to the simple boxing strategy described above. The set of first elements and the set of second elements are not boxed because of the lines pointing to elements of the sets.

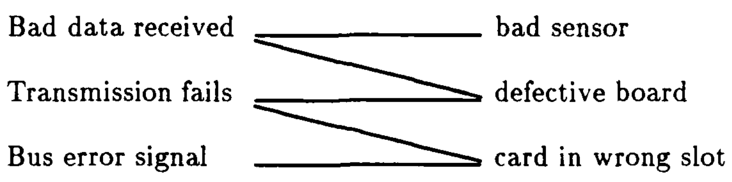


Figure 3. Line Display

Plot Chart

A plot chart visually represents a map $m : A \times B \mapsto C$. The set A is represented vertically and the set B horizontally (or vice versa), and placed offset so the values of $m(a, b)$ for every $a \in A$ and $b \in B$ can be displayed at the x position of b and the y position of a. See Figure 4 for an example.

The transformation has a similar form to that of the connecting line transformation but the annotations specifying layout are different:

$$\{\langle x,\ y\rangle \mapsto \textit{true} \mid p(x,\ y)\}$$
$$\longrightarrow$$
$$\lfloor \{x \mid p(x,\ y)\},\ \{y \mid p(x,\ y)\},\ \{\langle x,\ y\rangle \mapsto \textit{true} \mid p(x,\ y)\}\rfloor$$

For example, the specification

$$\{|\ \langle x,\ y\rangle \mapsto \textit{true} \mid \langle x,\ y\rangle \text{ in } \textit{Diagnosis-Relation}\ |\}$$

is transformed to

$$\begin{aligned}&\lfloor \{x \mid \langle x,\ y\rangle \text{ in } \textit{Diagnosis-Relation}\},\\ &\ \{y \mid \langle x,\ y\rangle \text{ in } \textit{Diagnosis-Relation}\},\\ &\ \{|\ \langle x,\ y\rangle \mapsto \textit{true} \mid \langle x,\ y\rangle \text{ in } \textit{Diagnosis-Relation}\ |\}\rfloor\end{aligned}$$

resulting in the display of Figure 4 (with "X" being used to represent the value "*true*").

	bad sensor	defective board	card in wrong slot
Bad data received	X	X	
Transmission fails		X	X
Bus error signal			X

Figure 4. Plot Chart Display

Redundancy Removal Rules

There are two kinds of redundancy removal rules: (a) one removes common subexpressions by merging, and (b) the second projects on a field of a relation to avoid repetition of that element of the relation. We give an example of (b) following; examples of (a) will be presented in the derivations of more complex tables below, where the motivation is clearer. Note that the programming language option of introducing a variable to hold the value of a common subexpression is not as simple in a visual representation and is not used in our system. Instead we perform a merging of the contexts in which the common subexpressions appear.

The rule that projects on the first element of a binary relation can be expressed as

$$\{\langle x,\ y\rangle \mid p(x,\ y)\}$$
$$\longrightarrow$$
$$\{\langle x,\ \{y \mid p(x,\ y)\}\rangle \mid p(x,\ y1)\}$$

For example, the specification

$$\{\mid \langle x,\ y\rangle \mid \langle x,\ y\rangle \text{ in } \mathit{Diagnosis\text{-}Relation} \mid\}$$

is transformed to

$$\{\langle x,\ \{y \mid \langle x,\ y\rangle \text{ in } \mathit{Diagnosis\text{-}Relation}\}\rangle$$
$$\mid \langle x,\ y1\rangle \text{ in } \mathit{Diagnosis\text{-}Relation}\}$$

resulting in the display of Figure 5.

Bad data received	defective board bad sensor
Transmission fails	card in wrong slot defective board
Bus error signal	card in wrong slot

Figure 5. Projection Display

Conditioning Rules

The conditioning rules for enabling applicability of redundancy removal rules are described in context in the derivations of more complex tables below. The only conditioning rule for enabling applicability of a visual representation rule that we require in this paper is the rule that converts a set to a boolean map (its characteristic function):

$$\{f(x) \mid p(x)\}$$
$$\longrightarrow$$
$$\{\mid f(x) \mapsto \mathit{true} \mid p(x) \mid\}$$

For example, the specification

$\{| \langle x, y\rangle \mid \langle x, y\rangle \text{ in } \textit{Diagnosis-Relation} |\}$

is transformed to

$\{| \langle x, y\rangle \mapsto \textit{true} \mid \langle x, y\rangle \text{ in } \textit{Diagnosis-Relation} |\}$

which enables the plot chart representation transformation.

Generator Structure

The display synthesizer creates a number of possible displays for the input specification. The generator performs a number of data reformulations and display style choices. For each successful combination of choices, a display program is synthesized. Compiling and executing the display program creates the screen or window output.

The generator actually consists of two generators acting in series. The first manipulates the specification into a variety of "equivalent" forms. For each form, the second generator cycles through possible choices of visual display styles. Both generators use top-down, successive refinement on the specification and its subparts. Each generator annotates the specification with various display directives. At the conclusion of the design generation the specification is fully annotated. Then the code synthesizer recursively descends through the specification, synthesizing display code in accordance with the directives.

The entirety of all the combinations of possible choices is called the search space. The specifics of the control strategies for the generators determine in what order the space is explored. One motive for extending our theory to include a selection component is to reduce the number of display possibilities that must be fully explored.

Since the control structure is independent of the actual transformations, more data reformulations and display styles can be incrementally added.

In the next three sections we look at derivations of more complex tables.

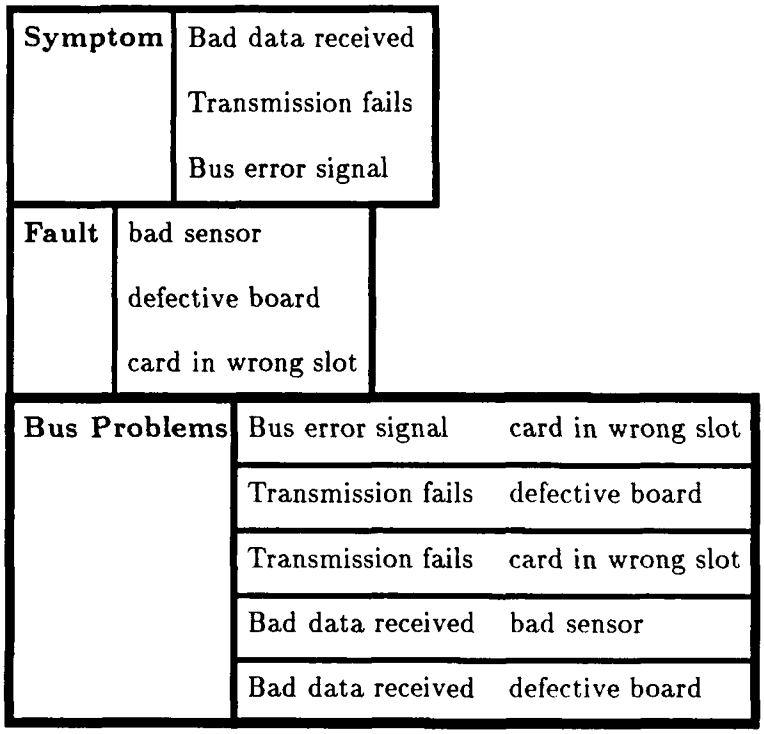
Bus Problem Diagnosis Table

In this example we consider the Diagnosis-Relation given above but with labels for the relation and its domain and range. The abstract specification for this is

$\lfloor\langle\text{"Symptom"}, \{x \mid \langle x, y\rangle \text{ in } \textit{Diagnosis-Relation}\}\rangle,$
$\langle\text{"Fault"}, \{y \mid \langle x, y\rangle \text{ in } \textit{Diagnosis-Relation}\}\rangle,$
$\langle\text{"Bus Problems"}, \{\langle x, y\rangle \mid \langle x, y\rangle \text{ in } \textit{Diagnosis-Relation}\}\rangle\rfloor.$

This display corresponding to this specification is shown in Figure 6.

Within each top-level tuple, the first and second elements are associated horizontally. The second element of each tuple is a set, the elements of which are associated vertically. The set elements for the third top-level tuple, which are themselves tuples, have their fields associated



Symptom	
	Bad data received
	Transmission fails
	Bus error signal

Fault	
	bad sensor
	defective board
	card in wrong slot

Bus Problems		
	Bus error signal	card in wrong slot
	Transmission fails	defective board
	Transmission fails	card in wrong slot
	Bad data received	bad sensor
	Bad data received	defective board

Figure 6. Diagnosis: Initial Display

horizontally. The top-level tuples are presented vertically one after the other as a default, although the specification does not demand this.

Expression Embedding

In the display in Figure 6 the domain and range are completely redundant because their elements all occur in the relation. To remove this redundancy the domain and range expressions cannot simply be deleted because the association with the domain and range labels would be lost. These associations can be maintained by replacing the elements in the relation by label-element pairs, after which the domain and range expressions can be safely omitted along with their label associations. The resulting specification is

$$\begin{array}{l}\langle\text{"Bus Problems"},\\ \quad\{\langle\langle\text{"Symptom"}, x\rangle, \langle\text{"Fault"}, y\rangle\rangle\\ \qquad \mid \langle x, y\rangle \text{ in } \textit{Diagnosis-Relation}\}\rangle\end{array}$$

and the resulting display is shown in Figure 7.

This transformation has traded off redundancy in domain and range elements with the redundancy of repeated label appearances. In some cases, this repetition might be useful. For example, if a program variable was present in multiple packages, it would be important to print the package name to be clear about which variable is being referenced. Or with some applications the repetition might be mandated—if the heading was a formal title such as the "The Honorable". Another option

Bus Problems

Symptom Bus error signal	**Fault** card in wrong slot
Symptom Transmission fails	**Fault** defective board
Symptom Transmission fails	**Fault** card in wrong slot
Symptom Bad data received	**Fault** bad sensor
Symptom Bad data received	**Fault** defective board

Figure 7. Diagnosis: After Embedding Domain and Range

is to encode the pairing using color, shape, or font, so less display space is used. This option is not yet implemented in our system. The next section shows how the redundancy can be removed.

Constant Extraction

We wish to avoid the redundancy of the repetition of "Symptom" and "Fault" introduced by the previous transformation. This repetition is reflected in the current specification as the constant strings being within a repeated part of the set comprehension:

$$\begin{array}{l}\langle\text{"Bus Problems"},\\ \quad\{\langle\langle\text{"Symptom"},\, x\rangle,\, \langle\text{"Fault"},\, y\rangle\rangle\\ \qquad \mid \langle x,\, y\rangle \text{ in } \textit{Diagnosis-Relation}\}\rangle.\end{array}$$

To avoid this repetition we must isolate the constants "Symptom" and "Fault" from the variables x and y. The conditioning rule for this purpose applies to the tuple

$$\langle\langle\text{"Symptom"},\, x\rangle,\, \langle\text{"Fault"},\, y\rangle\rangle$$

producing

$$\langle\langle\text{"Symptom"},\, \text{"Fault"}\rangle,\, \langle x,\, y\rangle\rangle.$$

This transformation is a transposition if we interpret the tuple of tuples as a matrix. However, visual representation does not guarantee this interpretation and other interpretations may not preserve the association of "Symptom" with x and "Fault" with y. Therefore, the rule adds an annotation to constrain the visual interpreter to make this association.

After this application the specification is

$$\begin{array}{l}\langle\text{"Bus Problems"},\\ \quad \{\langle\langle\text{"Symptom"}, \text{"Fault"}\rangle, \langle x, y\rangle\rangle\\ \qquad \mid \langle x, y\rangle \text{ in } \textit{Diagnosis-Relation}\}\rangle.\end{array}$$

With this form we may project the relation on its first element and thus avoid repetition of the labels:

$$\begin{array}{l}\langle\text{"Bus Problems"},\\ \quad \{\langle\langle\text{"Symptom"}, \text{"Fault"}\rangle, \{\langle x, y\rangle \mid \langle x, y\rangle \text{ in } \textit{Diagnosis-Relation}\}\rangle\\ \qquad \mid \langle x1, y1\rangle \text{ in } \textit{Diagnosis-Relation}\}\rangle.\end{array}$$

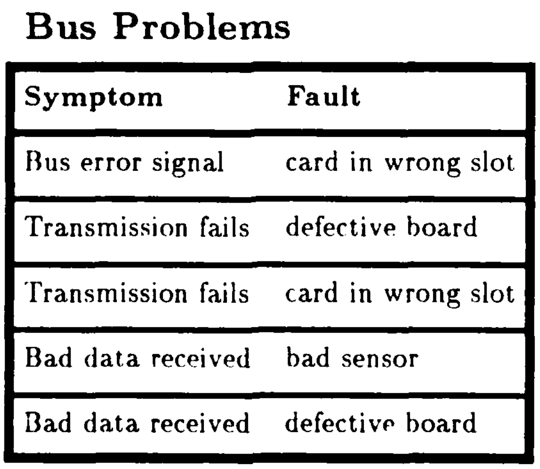
When Diagnosis-Relation is non-empty, this simplifies to

$$\begin{array}{l}\langle\text{"Bus Problems"},\\ \quad \{\langle\langle\text{"Symptom"}, \text{"Fault"}\rangle,\\ \quad\ \{\langle x, y\rangle \mid \langle x, y\rangle \text{ in } \textit{Diagnosis-Relation}\}\rangle\}\rangle.\end{array}$$

Displaying a singleton set is considered to be the same as displaying just the single element of the set. Therefore, this is the same as

$$\begin{array}{l}\langle\text{"Bus Problems"},\\ \quad \langle\langle\text{"Symptom"}, \text{"Fault"}\rangle,\\ \quad\ \{\langle x, y\rangle \mid \langle x, y\rangle \text{ in } \textit{Diagnosis-Relation}\}\rangle\rangle.\end{array}$$

The resulting display is shown in Figure 8.



Bus Problems

Symptom	Fault
Bus error signal	card in wrong slot
Transmission fails	defective board
Transmission fails	card in wrong slot
Bad data received	bad sensor
Bad data received	defective board

Figure 8. Diagnosis: After Removing Label Repetition

Avoiding Repetition Of Domain Elements

The remaining redundancies in this display are repetitions of the symptom and fault names. The repetition of the symptoms is remedied by projecting on the first element of the tuple

$\langle$"Bus Problems",
$\langle\langle$"Symptom", "Fault"$\rangle$,
$\{\langle x,\ \{y \mid \langle x,\ y\rangle$ in *Diagnosis-Relation*$\}\rangle$
$\mid \langle x,\ y1\rangle$ in *Diagnosis-Relation*$\}\rangle\rangle$

giving the display in Figure 9.

Bus Problems

Symptom	**Fault**
Bad data received	defective board bad sensor
Transmission fails	card in wrong slot defective board
Bus error signal	card in wrong slot

Figure 9. Diagnosis: After Domain Projection

This display still has a repetition of fault names but there is no simple transformation that can remove this repetition. Instead of projecting on the first element of the tuple to avoid repetition of symptoms, we could have projected on the second elements to avoid repetition of faults but then we would have had repetition of symptoms. We need to return to the original specification and use different transformations to avoid repetition of either.

Connecting Line Table

The original specification is

$\lfloor\langle$"Symptom", $\{x \mid \langle x,\ y\rangle$ in *Diagnosis-Relation*$\}\rangle$,
$\langle$"Fault", $\{y \mid \langle x,\ y\rangle$ in *Diagnosis-Relation*$\}\rangle$,
$\langle$"Bus Problems", $\{\langle x,\ y\rangle \mid \langle x,\ y\rangle$ in *Diagnosis-Relation*$\}\rangle\rfloor$.

The connecting line representation transformation applies to the final set comprehension giving

$\lfloor\langle$"Symptom", $\{x \mid \langle x,\ y\rangle$ in *Diagnosis-Relation*$\}\rangle$,
$\langle$"Fault", $\{y \mid \langle x,\ y\rangle$ in *Diagnosis-Relation*$\}\rangle$,
$\langle$"Bus Problems",
$\lfloor\{x \mid \langle x,\ y\rangle$ in *Diagnosis-Relation*$\}$,

$\{y \mid \langle x,\ y\rangle \text{ in } \textit{Diagnosis-Relation}\},$
$\{\langle x,\ y\rangle \mid \langle x,\ y\rangle \text{ in } \textit{Diagnosis-Relation}\}\rfloor\rangle\rfloor.$

Now a common subexpression merging rule applies. The rule detects the set of first elements from the Diagnosis-Relation is the same set it was asked to display in association with "Symptom". And the set of second elements has the same value as the set associated with "Fault". After identifying these common terms, the specification is rewritten to use each only once:

$\langle\text{"Bus Problems"},$
$\lfloor\langle\text{"Symptom"},\ \{x \mid \langle x,\ y\rangle \text{ in } \textit{Diagnosis-Relation}\}\rangle,$
$\langle\text{"Fault"},\ \{y \mid \langle x,\ y\rangle \text{ in } \textit{Diagnosis-Relation}\}\rangle,$
$\{\langle x,\ y\rangle \mid \langle x,\ y\rangle \text{ in } \textit{Diagnosis-Relation}\}\rfloor\rangle.$

The resulting display is shown in Figure 10.

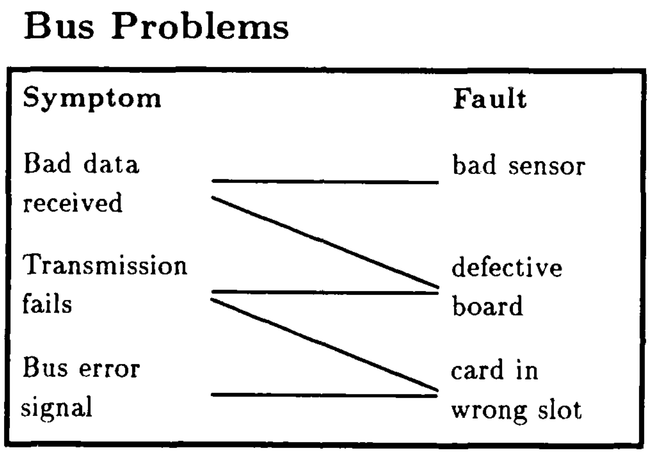


Figure 10. Diagnosis: Line Table

Plot Chart

Before the plot chart representation rule can apply, we must return to the original specification and then apply to the final set comprehension the conditioning rule that converts a set to a boolean map, giving

$\lfloor\langle\text{"Symptom"},\ \{x \mid \langle x,\ y\rangle \text{ in } \textit{Diagnosis-Relation}\}\rangle,$
$\langle\text{"Fault"},\ \{y \mid \langle x,\ y\rangle \text{ in } \textit{Diagnosis-Relation}\}\rangle,$
$\langle\text{"Bus Problems"},\ \{\mid \langle x,\ y\rangle \mapsto \textit{true} \mid \langle x,\ y\rangle \text{ in } \textit{Diagnosis-Relation} \mid\}\rangle\rfloor.$

The plot chart rule now applies to this map expression giving

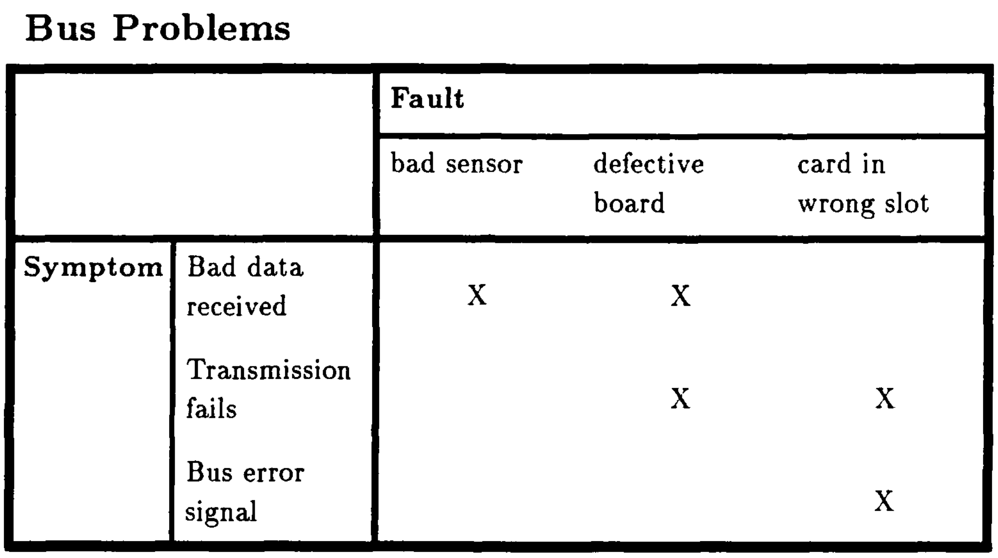
$\lfloor\langle\text{"Symptom"},\ \{x \mid \langle x,\ y\rangle \text{ in } \textit{Diagnosis-Relation}\}\rangle,$
$\langle\text{"Fault"},\ \{y \mid \langle x,\ y\rangle \text{ in } \textit{Diagnosis-Relation}\}\rangle,$
$\langle\text{"Bus Problems"},$
$\lfloor\{x \mid \langle x,\ y\rangle \text{ in } \textit{Diagnosis-Relation}\},$
$\{y \mid \langle x,\ y\rangle \text{ in } \textit{Diagnosis-Relation}\},$
$\{\mid \langle x,\ y\rangle \mapsto \textit{true} \mid \langle x,\ y\rangle \text{ in } \textit{Diagnosis-Relation} \mid\}\rfloor\rangle\rfloor.$

with appropriate annotations.

The same redundancy removal rule that merged the common subexpressions in the line table derivation also applies here, giving

$$\begin{aligned}&\langle \text{"Bus Problems"},\\ &\quad \lfloor \langle \text{"Symptom"}, \{x \mid \langle x, y\rangle \text{ in } \mathit{Diagnosis\text{-}Relation}\}\rangle,\\ &\quad \langle \text{"Fault"}, \{y \mid \langle x, y\rangle \text{ in } \mathit{Diagnosis\text{-}Relation}\}\rangle,\\ &\quad \{\mid \langle x, y\rangle \mid \langle x, y\rangle \text{ in } \mathit{Diagnosis\text{-}Relation} \mid\}\rfloor\rangle.\end{aligned}$$

The resulting display is shown in Figure 11.



Bus Problems

		Fault		
		bad sensor	defective board	card in wrong slot
Symptom	Bad data received	X	X	
	Transmission fails		X	X
	Bus error signal			X

Figure 11. Diagnosis: Plot Chart.

The entries in the map itself are given the sameness *plot-chart.* Referring to the map entries. $\langle x, y\rangle \mapsto z$ the meaning of plot-chart is that the z range values are to be positioned in the same column as the x value appears, and the same row as the y value. In order to do that, the system must keep track of the row and column coordinates as it prints the domain and range sets. The *Domain-Colm-Map* and *Range-Row-Map* are created by the transformation to hold this information.

After this transformation finishes the specification is annotated with the visual display choices. and the auxiliary data structures have been created. Then the synthesis of the actual display code begins. First, the code for printing the domain and range sets is synthesized. Within this code. the x and y coordinate values for the Domain-Colm-Map and Range-Row-Map will be stored. Then the code for the table entries is synthesized by the function Plot-Chart.

The important action taken by this synthesis step is to determine the coordinates for printing the value of z by setting the *current-colm* to be the column for x from the Domain-Colm-Map. and the *current-row* to be the row containing y from the Range-Row-Map.

Example: Multicolumn Table

This example shows the generation of a table with more than two columns. The data is a 4-ary relation:

Best-Options =
$\{\langle$"Replace Low Power RF Unit", 1.9, "30%", "9473A4"$\rangle$,
$\langle$"Repair LPRF/Computer Cable", 0.4, "20%", "N/A"$\rangle$,
$\langle$"Replace FCR Computer", 1.2, "50%", "9473A6"$\rangle$,
$\langle$"Check LPRF/Computer Cable", 0.4, "20%", "N/A"$\rangle\}$.

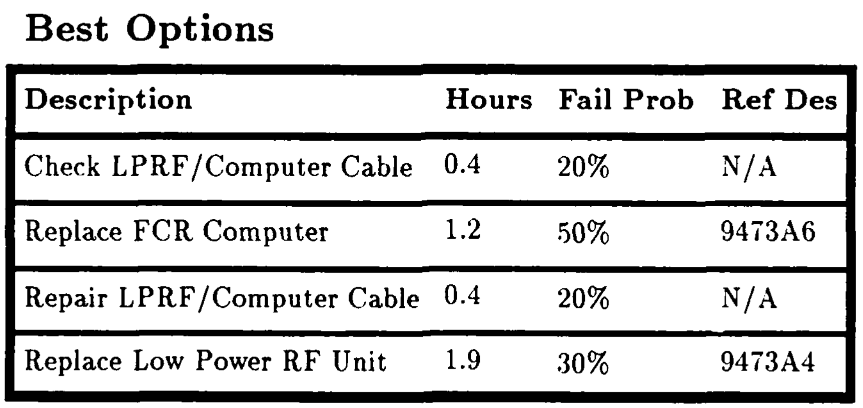
The initial description of the data to be displayed is

$\lfloor\langle$"Description", $\{W \mid \langle W, x, y, Z\rangle$ in *Best-Options*$\}\rangle$,
$\langle$"Hours", $\{x \mid \langle W, x, y, Z\rangle$ in *Best-Options*$\}\rangle$,
$\langle$"Fail Prob", $\{y \mid \langle W, x, y, Z\rangle$ in *Best-Options*$\}\rangle$,
$\langle$"Ref Des", $\{Z \mid \langle W, x, y, Z\rangle$ in *Best-Options*$\}\rangle$,
$\langle$"Best Options", $\{\langle W, x, y, Z\rangle \mid \langle W, x, y, Z\rangle$ in *Best-Options*$\}\rangle\rfloor$.

The same initial transformations as for the diagnosis example apply giving

$\langle$"Best Options",
$\langle\langle$"Description", "Hours", "Fail Prob", "Ref Des"$\rangle$,
$\{\langle W, x, y, Z\rangle \mid \langle W, x, y, Z\rangle$ in *Best-Options*$\}\rangle\rangle$

which produces the display in Figure 12.



Best Options

Description	Hours	Fail Prob	Ref Des
Check LPRF/Computer Cable	0.4	20%	N/A
Replace FCR Computer	1.2	50%	9473A6
Repair LPRF/Computer Cable	0.4	20%	N/A
Replace Low Power RF Unit	1.9	30%	9473A4

Figure 12. Multi-Column Table

Because the relation is not binary, fewer of the other transformations apply and none lead to any other interesting displays.

Example: Complex Plot Chart

This example shows how plot charts sharing a common domain may be combined. The original specification is

$$\begin{array}{l}
\lfloor\langle\text{"Faults"},\ \{F \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\}\rangle,\\
\ \langle\text{"Symptoms"},\ \{S \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\}\rangle,\\
\ \langle\text{"Tests"},\ \{TS \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\}\rangle,\\
\ \langle\text{"Repairs"},\ \{R \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\}\rangle,\\
\ \langle\text{"Fault Information"},\\
\quad \{\langle F,\ S\rangle \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\},\\
\quad \{\langle F,\ TS\rangle \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\},\\
\quad \{\langle F,\ R\rangle \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\}\rangle\rfloor.
\end{array}$$

There are many displays the table generator produces for this specification. The only ones introducing new ideas are the ones combining plot charts for the relations, so only the applications of the common subexpression merging rule leading to the combined charts are shown.

After two of the relations have been transformed into maps and represented as plot charts, and one of them has had its domain and range embedded in it, we have

$$\begin{array}{l}
\lfloor\langle\text{"Symptoms"},\ \{S \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\}\rangle,\\
\ \langle\text{"Tests"},\ \{TS \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\}\rangle,\\
\ \langle\text{"Fault Information"},\\
\quad \{\langle F,\ S\rangle \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\},\\
\quad \lfloor\{F \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\},\\
\quad\ \ \{TS \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\},\\
\quad\ \ \{\mid \langle F,\ TS\rangle \mapsto \textit{true} \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info} \mid\}\rfloor,\\
\quad \lfloor\langle\text{"Faults"},\ \{F \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\}\rangle,\\
\quad\ \ \langle\text{"Repairs"},\ \{R \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\}\rangle,\\
\quad\ \ \{\mid \langle F,\ R\rangle \mapsto \textit{true} \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info} \mid\}\rfloor\rangle\rfloor.
\end{array}$$

The range is embedded as in previous derivations but the repeated domains require a more subtle merging so as not to lose information. The result is

$$\begin{array}{l}
\lfloor\langle\text{"Symptoms"},\ \{S \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\}\rangle,\\
\ \langle\text{"Fault Information"},\\
\quad \{\langle F,\ S\rangle \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\},\\
\quad \lfloor\langle\text{"Faults"},\ \{F \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\}\rangle,\\
\quad\ \ \langle\langle\text{"Tests"},\ \{TS \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\}\rangle,\\
\quad\ \ \ \ \langle\text{"Repairs"},\ \{R \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\}\rangle\rangle,\\
\quad\ \ \lfloor\{\mid \langle F,\ TS\rangle \mapsto \textit{true} \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info} \mid\},\\
\quad\ \ \ \ \{\mid \langle F,\ R\rangle \mapsto \textit{true} \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info} \mid\}\rfloor\rfloor\rangle\rfloor.
\end{array}$$

After transforming the third relation into a map and representing it as a plot chart, the same common subexpression merging process produces

$$\begin{array}{l}
\langle\text{"Fault Information"},\\
\quad \langle\langle\text{"Faults"},\\
\quad\ \ \ \ \{F \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\}\rangle,\\
\quad\ \ \langle\langle\text{"Symptoms"},\ \{S \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\}\rangle,\\
\quad\ \ \ \ \langle\langle\text{"Tests"},\ \{TS \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\}\rangle,\\
\quad\ \ \ \ \ \ \langle\text{"Repairs"},\ \{R \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info}\}\rangle\rangle\rangle\rangle,\\
\quad \lfloor\{\mid \langle F,\ S\rangle \mapsto \textit{true} \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info} \mid\},\\
\quad\ \ \{\mid \langle F,\ TS\rangle \mapsto \textit{true} \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info} \mid\},\\
\quad\ \ \{\mid \langle F,\ R\rangle \mapsto \textit{true} \mid \langle F,\ S,\ TS,\ R\rangle \text{ in } \textit{Fault-Info} \mid\}\rfloor\rangle\rangle.
\end{array}$$

which gives the display in Figure 13. The simple boxing strategy used does not work very well here because it does not recognize an incidental nesting of range labels and sets. There is a box missing from the entire table because only three levels of boxes are allowed.

Fault Information

		Symptoms		**Tests**		**Repairs**		
		RDR-595	RDR-018	9461BIT	9461AW1	R_FCRCom	R_LPRF	RP_CabCo
Faults	F018A	X	X	X		X		
	F018B	X	X	X			X	
	F018C	X	X	X	X			X

Figure 13. Combined Plot Chart

IV. DIAGRAM SYNTHESIS

This section describes the prototype diagram synthesis system we have developed. The overall goal we are pursuing is a coherent and uniform presentation of functional data. In contrast to table generation, the principle problem is constructing a pleasing and appropriate layout of all the diagram components and consonant drawing of any connecting arcs.

The first subsection below presents a brief overview of the issues involved in diagram synthesis and describes our general approach. The next subsection discusses design foundations of the present system; the following one examines in more detail the layout strategies currently implemented. Several example diagrams synthesized by the system are included in Section IV as well.

Overview

As with table synthesis, we seek to represent functional similarities and interconnections as visual ones. Since the diagrams we construct are presently limited to boxes and lines (the lines represent functional or dependency-based connections), this goal primarily amounts to arranging and rendering the diagram's components so that such similarities are immediately conveyed.

The Issues

In general, we consider the problem of producing a pleasing diagram layout as one of *constraint satisfaction*, as in (Voigt & Tong, 1989) or (Braudaway, 1989). A number of issues that may be viewed as constraints enter into the construction of a graphical representation for relational data. Chief among these are overall organization and space management—the resulting diagram must provide an easily assimilated presentation of the intended interrelationships, and will normally be required to fit within the boundaries of some predetermined space. These basic goals can often conflict, since a natural way to emphasize functional distinction and also reduce visual complexity is to space elements further apart.

Another way to emphasize similarity or distinction is through the use of colors, shading, and font selection for component labels. We do not presently make heavy use of different colors and fonts, though we do support several varieties of shading to indicate the current maintenance status of a given component. These may be selected and modified directly through pop-up menu selections within the user interface.

For the IMIS application, we need to synthesize diagram presentation packages that can be executed through different graphics utilities on a variety of display hardware. Presentation on a hand-carried portable, for instance, will impose more severe size constraints than on a large-screen work-station. As another example, presentation on paper media will generally not (yet) involve the use of color and may also restrict the availability of different fonts.

General Approach

We expect to receive abstract relational input, without explicit formatting or layout information, plus high-level forms of guidance regarding desired properties of the diagram to be constructed. Such guidance can simply be a selection and/or ordering of particular synthesis strategies to apply. Layout constraints could also take the form of logic-based rules extracted from the semantics of the entities being represented.

Human factor constraints provide another class of layout guidance (Ding & Mateti, 1990), (Gunning, 1987). These can range from limits on the minimum text fonts or on the number of different elements presented, to color selections and maximum allowed angles within a link (connection) path. Target display hardware capabilities or limitations must also be taken into account. These might be thought of as *machine-factor* constraints.

We construct a given layout through incremental assignment of various positioning attributes to diagram objects. This positioning is described according to a geometric model defined within the REFINE^{TM} knowledge base. A number of coordinate and direction transformations are supported.

Positioning of all diagram objects is expressed through relative coordinates until the entire diagram has been constructed. Then absolute positions can be assigned, based upon the size of the entire diagram and its placement within a surrounding screen or page coordinate system. Finally, connecting links will be laid out to complete the diagram.

A backtracking control structure cycles through different layout possibilities, pausing to display each completed diagram which passes all current constraints. The user can stop the generation sequence at any time to edit the current diagram, adding or deleting components and their interconnections with the mouse. A new generation sequence can then be invoked, with new constraints and selection of different strategies to apply, if so desired.

Design Foundations

Generation and Selection

As discussed in Section II, the diagram synthesis process may be viewed as comprising a generation and a selection part. The generator produces a series of box arrangements and connecting line placements from which the selector chooses a subset to be presented as complete drawings. At present, the selector makes its choices only on the basis of some simple measures of connecting link (line) complexity and aesthetics (Ding & Mateti, 1990).

Similar to the table generation control structure outlined in Section III, the diagram generation control structure also consists of two generators acting in series. In diagram synthesis, the first generator combines transformations of the initial input relation and subsequent construction of a layout plan. A given set of transformations and plan construction rules corresponds to a particular layout *strategy.*

For each strategy applied, the second generator recursively steps through the layout plan, producing a number of potential placements for each diagram component. Whenever the sequence of placement choices leads to a plausible total arrangement,[2] derivation and positioning of line segments to represent component linkage completes the display instance. The current implementation constructs the single "best" set of link lines for a given arrangement, again following the layout plan and backtracking to revise earlier choices when it encounters a conflict (such as a line crossing or an attempt to draw through a component box).

Bounding Box Abstraction

Both the diagram and table synthesis systems use a hierarchy of *bounding boxes* to allocate and manage screen real estate. Each box represents a rectangular region surrounding one or more components of a diagram (or table).

The box hierarchy forms a tree, in which a box surrounding the entire diagram corresponds to the root and individual components form the leaves. Boxes around groups of components correspond to inner branches, and permit diagram subgroups and their components to be laid out independently. Such groupings not only enforce visual association among similar objects but help combat combinatorial explosion as well. The dashed outlines in Figure 14 show the bounding boxes in a simple diagram.

Currently, a bounding box hierarchy is created just after the layout plan has been constructed. This fixed hierarchy is used for all the box placements generated from the associated layout plan. In future versions, we envision failures to satisfy top-level size or link-path constraints will lead to changes in the placement of some diagram components and attendant inner restructuring of the box hierarchy.

[2]For now, a plausible arrangement is just one in which no boxes overlap.

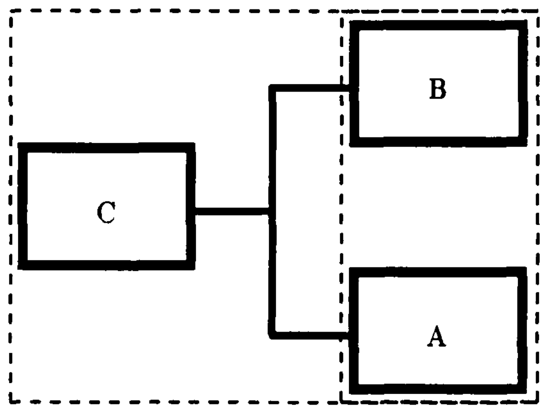


Figure 14. Bounding Box Hierarchy in a Simple Diagram.

Geometry Model

Individual diagram components and subgroups are positioned relative to each other through the use of a two-dimensional geometry that builds upon the box abstraction. The geometry permits a wide range of possible alignments along directed offset-vectors of arbitrary length. Figure 15 illustrates the available alignment points and offset-vectors supported.

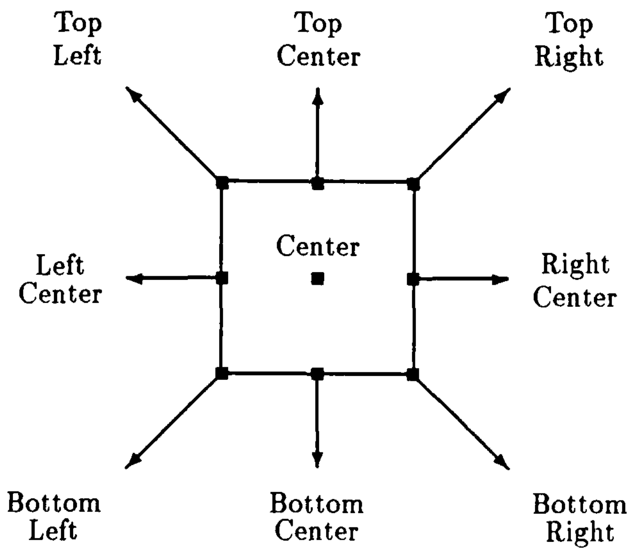


Figure 15. Offset Directions and Box Alignment Points.

In the most general case, a box is positioned with respect to another box through specification of an offset-vector, its magnitude, and a pair of alignment points, one for each box.

For example, positioning a box some positive distance *d* directly to the right of an already placed box corresponds to specifying a *Right* offset-vector of magnitude *d* and the alignment pair (*Right Center*, *Left Center*). That is, the new box will be positioned such that its left-center point will lie d units to the right of the right-center point of the already placed box. This is precisely the alignment illustrated in Figure 14, in which the bounding box containing B and A has been positioned with respect to component box C.

In the future, we expect to use the geometry model to enforce additional placement constraints specified by the user, such as "A *Left-Of* C," or "A *Above* B." Such positioning relations will also be derived from incorporation of *flow* information, classification of components as either input or output elements, and so forth.

Initial Layout Strategies

This subsection discusses two of the initial layout strategies we have developed and implemented. As introduced above, a layout *strategy* encompasses (a) transformations on the original input relation, (b) construction and use of a layout *plan*, and (c) constraints and/or preference orderings on the particular box placements generated within the plan. We begin with a brief description of the input transformations, and conclude with a look at how the connecting links are constructed.

Connection-based Input Transformations

The primary input to the diagram system is simply a relation specifying the links or connections existing between the various components. More precisely, the current input formalism is a set S of sets $s_1, \ldots, s_n$, where each s_i is a set of diagram components and

$$\{x, y\} \subset s_i \text{ for some } s_i \in S \Longrightarrow \textit{connected}(x, y).$$

For example, the input specification for the diagram in Figure 14 was just $\{\,\{A, B, C\}\,\}$. As a notational convenience, the text strings labels for each component may be specified separately, as a map, say $\{|\ A \rightarrow \text{"alpha"},\ B \rightarrow \text{"beta"},\ C \rightarrow \text{"gamma"}\ |\}$.

In the absence of additional information, individual components are collected into subgroups solely on the basis of their connectedness. We apply several transformations to the original input relation, producing a standard form that isolates, for example, the most or least connected elements, and collects remaining elements with similar connectivity into such subgroups. These subgroups will eventually be placed within bounding boxes and laid out in a uniform way.

The transformed relation and its internal subgroups are also sorted in terms of the total number of connections for each element and subgroup. Depending on the current layout strategy, these sorts may place more connected components before less connected ones, or vice-versa.

Orthogonal "Center-Out" Strategy

The basic idea in the *Center-Out* strategy is to first fix a position for the most connected component(s) and then distribute the remaining diagram elements around it. This notion is applied recursively to the subgroups of the diagram as well. The general presentation theory at work here is that placing the item with the largest number of emanating links first, in the center, will generally lead to fewer conflicts in eventual link-path construction.

For example, the input relation

$$\{\ \{fcr, cw\},\ \{fcc, re, co, cw\},\ \{rc, ss, tg, cw\}\ \},$$

with associated text labels map

$$\left\{ \left| \begin{array}{ll} rc \rightarrow \text{"Radar Control Panel"}, & fcc \rightarrow \text{"FCC"}, \\ tg \rightarrow \text{"Throttle Grip"}, & re \rightarrow \text{"REO"}, \\ ss \rightarrow \text{"Side Stick Controller"}, & co \rightarrow \text{"Cool"}, \\ cw \rightarrow \text{"Control Wiring"}, & fcr \rightarrow \text{"FCR"} \end{array} \right| \right\},$$

which represents components of a radar control subsystem, finds a number of pleasing layouts under this strategy.

After application of the input transformations, the relation has the form

$$[\ \langle\ [cw],\ [\ [tg, ss, rc],\ [co, re, fcc],\ [fcr]\ ]\ \rangle\ ],$$

in which *cw* has been isolated as the most connected component. and the three subgroups [*tg*, *ss*, *rc*], [*co*, *re*, *fcc*], and [*fcr*] have been collected for positioning around it. Accordingly, the layout plan is

$$[\ \langle\ [[cw]],\ [\,]\ \rangle,\ \langle\ [[tg, ss, rc],\ [co, re, fcc],\ [fcr]],\ [cw]\ \rangle\ ].$$

Here each tuple $\langle a, b\rangle$ indicates a collection of items (a) to be laid out with respect to one or more other items (b). The central component ([*cw*]) may be placed arbitrarily, that is, laid out with respect to nothing ([ ]). Figure 16 shows one of the layouts synthesized from this plan; dashed outlines again show the bounding boxes for the subgroups.

All the alignments in Figure 16 are center-based (on the *cw*). In this layout, the directions [*Left*, *Right*, *Above*] have been chosen for the respective subgroups. When the right-hand element of a layout plan tuple is a collection of components that lie within different bounding boxes, a *ghost* bounding box is automatically constructed to enclose them. This box can then be used to fix a relative position for the left-hand collection(s) of the tuple.

The two subgroup triples have each been laid out orthogonally to the offset-vector for the bounding box that contains them. This method is standard for both layout strategies.

Breadth-First "Linear" Strategy

The breadth-first strategy essentially tries to lay out things in a straight line. Relaxation of positioning constraints to generate additional layouts permits a number of exceptions to this; in general, the strategy prefers to move from left to right (or top to bottom), placing components and subgroups as it goes. In this strategy, the sorting predicates are reversed from those described above; the layout starts with the least connected item(s) and progresses to the more connected ones.

For example, given the input relation

$$\{\ \{w, x\},\ \{a, w\},\ \{l, w\},\ \{m, r, f, c\},\ \{l, d, x, c\}\ ,\}$$

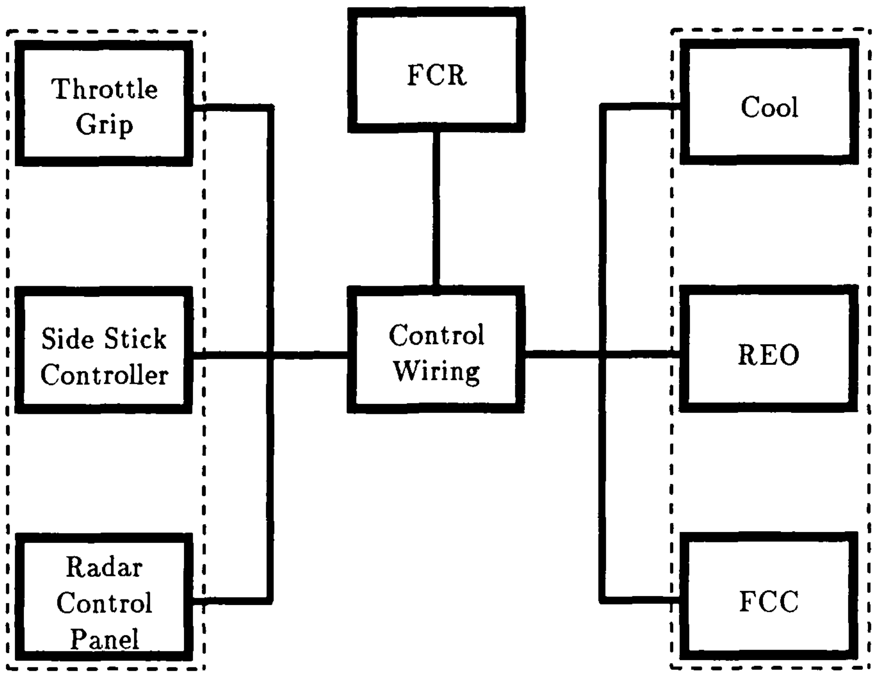


Figure 16. Center-Out Layout Example.

with text labels

$$\left\{ \left| \begin{array}{lll} l \rightarrow \text{"LPRF"}, & f \rightarrow \text{"FCC"}, & w \rightarrow \text{"Waveguide"}, \\ d \rightarrow \text{"DSP"}, & x \rightarrow \text{"Xmtr"}, & r \rightarrow \text{"Radar Panel"}, \\ m \rightarrow \text{"MUX"}, & a \rightarrow \text{"Antenna"}, & c \rightarrow \text{"FCR Computer"} \end{array} \right| \right\},$$

representing a radar signal processing subsystem, the initial input transformations produce the form

$$[\langle [a], [[w]] \rangle, \langle [w], [[l], [x]] \rangle, \langle [l, x], [[d, c]] \rangle, \langle [c], [[f, r, m]] \rangle],$$

in which the a component has been selected as a starting point. The layout plan is then

$$[\langle [[a]], [\,] \rangle, \langle [[w]], [a] \rangle, \langle [[l], [x]], [w] \rangle, \langle [[d, c]], [l, x] \rangle, \langle [[f, r, m]], [c] \rangle].$$

The meaning of the layout tuples is as above, and the strategy again works from left to right through the plan. Figure 17 illustrates one of the resulting layouts.

Here the bounding box containing the d and c components has been laid out with respect to a ghost box surrounding the l and x components, as specified by the $\langle [[d, c]], [l, x] \rangle$ tuple shown in the plan. Since l and x have been placed at the far ends of their bounding box, this ghost box turns out to be the same as their bounding box.

Link Construction

Note that in the above diagrams, several links have been joined to indicate common connections. When an item A is linked to each of a collection of items $B_1, \ldots, B_n$, and each of B_j and B_k

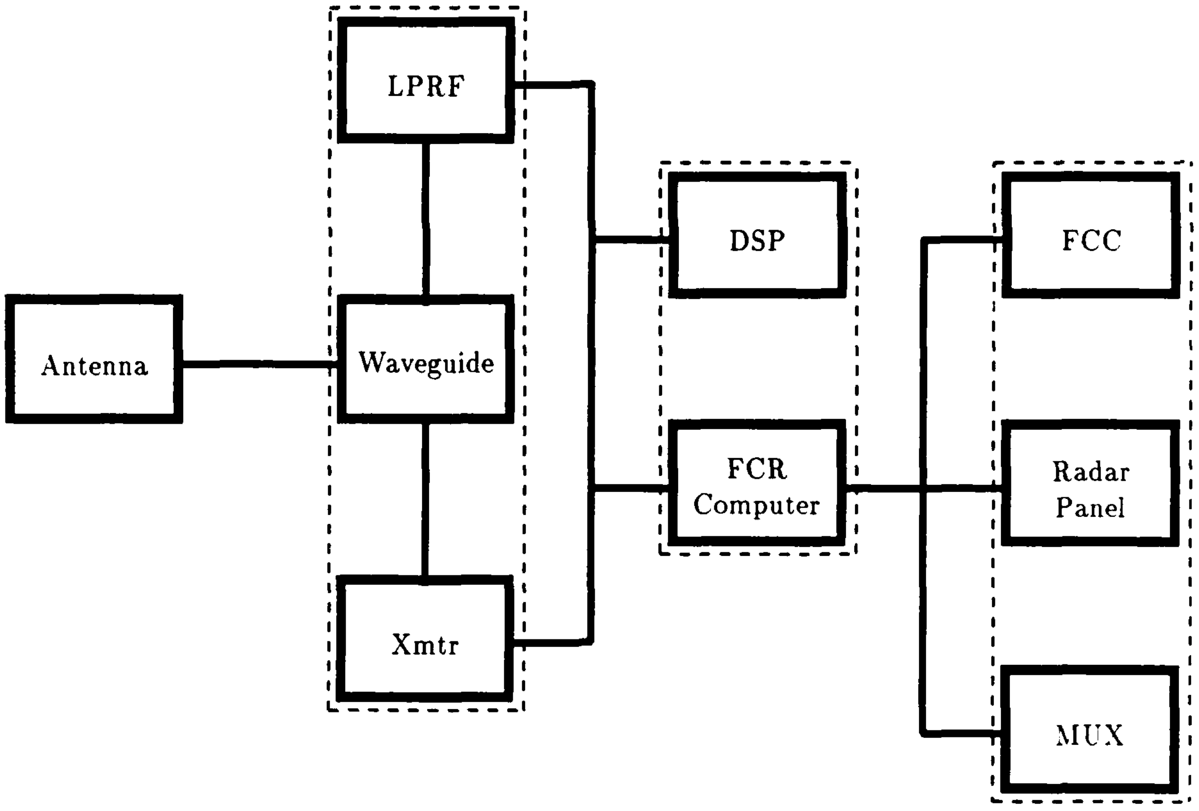


Figure 17. Breadth-First Layout Example.

$1 \leq j, k \leq n$ are linked to each other as well, the system will generally place all the B_i in a bounding box BB and create a high-level link object from A to BB. A *nexus point* will then be created and eventually positioned somewhere between A and BB as a target for $n + 1$ simple links—one from A and n from the B_i.

In drawing simple links the prototype system first attempts to use a straight horizontal or vertical line. If that will not work (because of interference from component boxes or other already laid out links), it then attempts to construct a suitable link by joining a horizontal and vertical segment as an *elbow* link. Figure 18 shows the elbow links drawn to a "Signal Processing" component added to the diagram of Figure 16.

The present restriction to either orthogonal lines or single elbows means there are relatively simple layouts and interconnections that cannot be drawn entirely with "approved" links. For instance, removing the new "Signal Processing" component and instead connecting the "FCC" and "Radar Control Panel" directly to each other presents such a case. In this instance, we would probably like to find a point in the neighborhood of the added component and use it, like the nexis points discussed above, as an intermediate target for the two elbows.

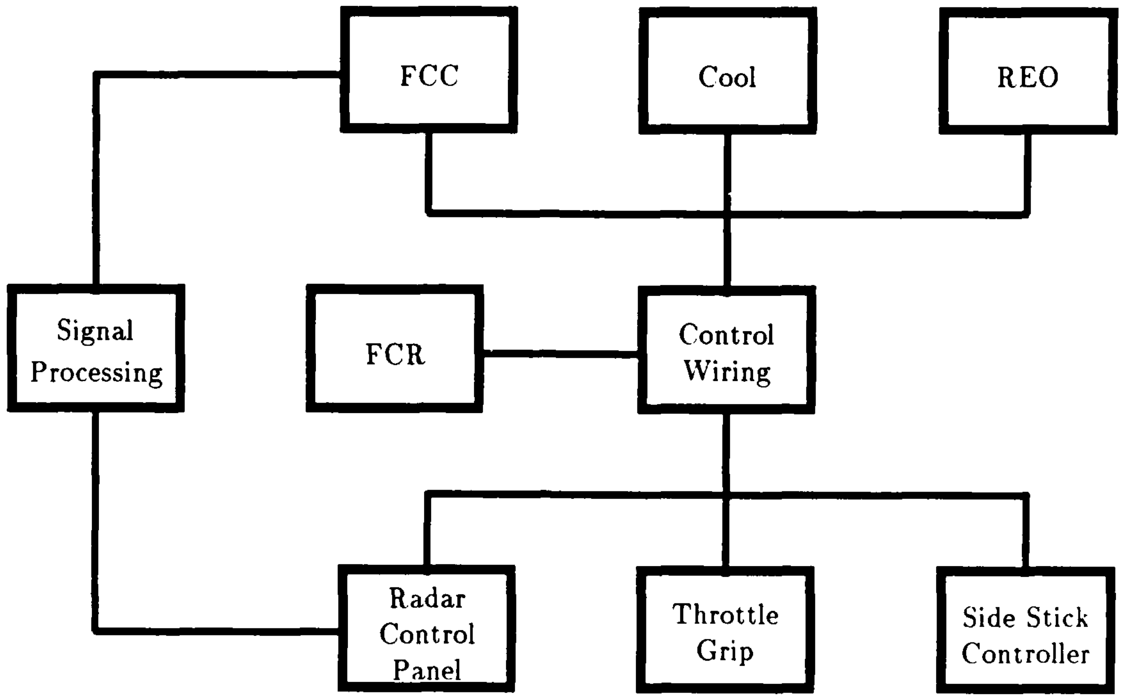


Figure 18. Link Paths Example.

V. FURTHER WORK

The work on the table generator is now fairly mature in that it can generate at least the basic shape of all the different kinds of tables we have seen. The diagram generator is less mature: we need to experiment both with additional high-level structuring strategies and layout strategies in order to generate additional interesting displays. Both subsystems require the addition of a more sophisticated selector theory and better customization of formatting details as discussed below.

Performance Measures

In order to reduce the number of plausible displays, we want to develop a measure based on the structural properties of the data that could indicate when a synthesis branch would not be worth pursuing. The measure will be concerned with properties such as the size of domain or range of relations. Is the mapping one-to-one or almost one-to-one? How many duplicate values are there among domain or range sets? For example, if many values were repeated, the synthesis branch that kept each repetition might not be worth exploring because of the amount of space each display would require. Scrolling of displays offers another option for extending the usable space that may or may not be appropriate within a given situation.

Usage Measures

Another metric that could help select among possible displays is a model of the amount of work it takes a user to answer a given question using a certain table. What actions are required, such as reading across a row, looking up something in a legend, or comparing two values? Can this work be quantified?

A similar metric could apply to the data structure itself before rendering decisions. The issues would be how many levels deep in the data structure one needed to look, what the average or worse case access times would be, and so on.

Usage maximization becomes a multiple objective optimization problem when more than one question is to be answered from the same table. When does it become beneficial to duplicate the data, perhaps in a variety of tables each suited for a particular need? When does the cost of the redundancy outweigh the convenience?

Variety of Displays

A natural extension of this theory is to generate more styles of displays. The work by Beach (Beach, 1985) on composing tables could be included in our rendering level. His description language for tabular displays includes concepts of fixed or unequal width columns, headings, underlining, centering, or right or left justification, etc.

Examples that could be generated by additional rendering capabilities and data structure reformulations are indexes of technical books (using a variety of fonts and underlining to indicate the semantics of the page reference), prettyprinting, and tree or graph displays.

Human Factors

A quality display synthesizer should use information about human perceptual limits to prune the space of possible displays, and later to choose certain presentations. For example, the human factors model should know what adjacent colors or shadings can be distinguished by a human viewer. Guidelines about the amount of white space needed to enhance readability would be useful in determining row spacing, column widths, and the amount of text to include on one page.

J. Mackinlay (1986) summarizes the effectiveness of different visual characteristics such as position, length, texture, area, density, shape, volume, color hue, and so on. with respect to the perception of three classes of information. The three classes are (a) quantitative information (values chosen from some range), (b) ordinal information (ordered values but not necessarily numeric, as in Monday, Tuesday,...), and (c) nominal information (an unordered collection as in Argentina, Peru, Brazil,...). For any of the three classes, people most accurately perceive a position-based encoding. People are not good at comparing areas: using circles of different diameters works acceptably to encode a range of values but it is difficult for people to perceive the encoding for ordinal information unless the step size between the areas is great enough. And when used for encoding nominal information, the difference in areas will cause people to perceive an ordering on the items that is not meant.

E. R. Tufte (1983) similarly disputes the use of area as an effective representation. He cites studies showing how poorly people compare areas. For example, people perform poorly when given an initial circle, and when asked to choose from other circles the one containing twice the area of the given circle. This human factors knowledge would be useful to make the display synthesizer refrain from encoding the quantity to compared as area.

A notion of intent could also guide the visual representation choices. Excepting mandated forms, such as accounting statements, the user wants to tell or see a story in the data.

Domain Specific Graphics

Each of the major professions has its own language, which includes graphical elements as well as words. Within a professional area, certain applications may have their own visual vocabulary. Some visual conventions are specific to certain domains—for example the use of red and black for loss or gain in accounting, and replacing trailing "000" in annual reports by one statement. "amounts are in thousands of dollars."

The domain-specific knowledge should customize the display synthesizer. The conventions of a field might not be used for a given display but care should be taken not to violate them either. It would mislead a reader of a financial display if the color red represented credit entries.

For some domains, standard presentations of information are used. In the project management area, some standard forms are PERT charts, calendars, and GANTT charts. MacProject has seven types of charts derivable from planning information. Our synthesis techniques should be able to synthesize these charts. However, after that capability is demonstrated it will be more efficient to save the display routine rather than recreate it each time.

Other professional areas with their own visual languages include computer-aided design/computer-aided manufacturing, program development (such as SADT or Jackson methods), decision support systems, decision trees, architecture, and each of the physical engineering fields.

Multiple Displays, Pages

Placement of figures on a page or multiple pages needs to be considered when displaying more than one relation. If the relations are analogous, (e.g., showing the same functional information at different points in time), then the same style of presentation should be used for each. The synthesis process must somehow communicate among or coordinate these displays.

If a large relation requires more than a one-page display, then some information should be repeated on each page in order to make the display locally (to the page) comprehensible. For example, the row and column headings of a large chart are often duplicated on each page. We need to decide what information should be duplicated and how to position it in order for the reader to grasp the continuity of the relation.

Thus, we will need to explore the requirement for an abstract constraint solver that resolves symbolic descriptions of the visual display properties. For example, the constraints must express the notions of adjacency and overlapping. Another type of constraint restricts the visual representation choices. An existing example is the rule alternating between vertical and horizontal alignment.

In general, making a visual representation choice at one place in the data structure might rule out some other visual choices elsewhere in the data structure.

Our transformational capabilities might be able to help fine-tune the synthesized displays for specific needs. For example, how can a slightly oversized display be transformed to fit within a page? What information do we choose to omit, abbreviate, move, or shrink? What transformations could be used to help a display that doesn't work?

In summary, we are solving *design* problems. Constraint-based layout of displays has the same challenges as designing offices, work areas, engines, electrical wiring, or circuit board layouts. The techniques being developed for constraint-based programming will contribute to the design problem definition and solution.

Interactive Graphics

There is significant literature in the proceedings of the Association of Computing Machinery (ACM) annual SIGGRAPH conference and the ACM Transactions on Graphics journal entitled User Interface Management Systems (Olsen, 1987). Areas of active research include specification languages for designing interactive displays, and toolboxes for implementing them.

User interaction during the synthesis of the graphics display could help reduce the search space by proceeding more directly to the style of display a user needs. Possible directives that could be given by the user as she or he views a partially synthesized display include instructions to swap axes, reorder columns, move labels, or choose an undo option. However, we doubt there is a natural direct manipulation interface that would achieve the deep restructuring performed by our reformulation steps. That means the user may be able to direct the search within a certain branch of the tree but not jump across branches.

Related Work

The work most similar to ours is that of Mackinlay (1986) whose viewpoint we share to a large extent. His work explores the use of graphical languages for encoding relational information. The use of visual associations used in our paper can be considered to be an instance of this idea, specialized for our particular applications. Only a few of the kinds of tables and diagrams presented in our paper are considered by Mackinlay: offset plots, and diagrams with very simple layout. Another difference is Mackinlay's emphasis on effectiveness of displays (our "selector") as opposed to our emphasis on a fine-grained generator of displays.

Several researchers have developed automated layout systems for tree-structured diagrams, (Wetherell & Shannon, 1979), (Reingold & Tilford, 1981), (Robins, 1987), and (Moser, 1990). Relatively little work has been done automating the layout of general graph structures. An algorithm for laying out Entity/Relationship diagrams is described (Tamassia, et. al., 1983). It uses a fixed reference grid for positioning entities, performs several graph-theoretic transformations, and solves a linear programming problem in attempting to minimize edge crossings and total edge path length. Their layouts are more restricted than ours and do not include any subgrouping mechanisms. We have not as yet applied any formal minimization techniques, such as linear programming, to our link construction methods.

Tichy and Newbery (1987) discuss a knowledge-based graphical editor with capabilities similar to ours for adding/deleting components and edges, highlighting or shading individual components, and so on. They also concentrate on minimization of edge crossings and are investigating mechanisms for accommodating user specification of layout style, such as component grouping, bounding box abstractions, and relative positioning. Our system already uses these mechanisms; we are also looking into the construction of multilevel diagrams and facilities for zooming in and out between the levels.

There are many commercial word processing, spreadsheet, and database programs with table-displaying components. We know of none concerned with the kinds of restructuring of displays as described in this paper.

The October 1989 IEEE Computer Special Issue on Visualization in Computing (IEEE Computer, 1989) describes recent work on generating visualizations for use in computing environments.

VI. CONCLUSIONS

We have developed a generative theory for the automated synthesis of tabular and diagrammatic displays. The prototype based on this theory demonstrates that a small number of design principles can succeed in creating a variety of interesting and useful displays. The success of this approach raises hopes for increasing the range of problems tackled by automated programming to include useful and creative input/output capabilities. Continuation of this effort could help reduce costs of porting programs to new hardware by replacing handcoding with resynthesis of the display code.

The theory can be extended. New data structure reformulations and new visual display possibilities can be added to the existing framework. Application-specific display knowledge could be added. A theory for the selector component, and performance and usage measures will complement the generator.

In summary, parallels exist between display synthesis and program synthesis that have not yet been mined by our research effort; the path begun here seems to lie in a rich vein.

REFERENCES

Beach, R. J. (1985). Setting Tables and Illustrations with Style (CSL-85-3). Palo Alto, CA: Xerox Palo Alto Research Center.

Braudaway, W., and Tong, C. (1989). Automated Synthesis of Constrained Generators Working Paper No. 118. Rutgers: Rutgers University, Computer Science Department.

Ding, C., and Mateti, P. (1990). A framework for the automated drawing of data structure diagrams. IEEE Transactions on Software Engineering, 16(5), 543–557.

Gunning, D. (1987). A General Framework for Describing Human-Computer Information Systems. Unpublished manuscript.

Shriver, B. (1989) IEEE Computer, 22(10) Special Issue on Visualization in Computing.

Kotik, G. B., Rockmore, A. J., and Smith, D. R. (1986, August). Use of $REFINE^{TM}$ for knowledge-based software development. In Robinson, G. S. (Ed) Westex '86 Conference. (Los Angeles, CA). (pp 106–110). New York: IEEE Press.

Mackinlay, J. (1986). Automating the design of graphical presentations of relational information. ACM Transactions on Graphics, 5(4), 110–141.

Moser, L. E. (1990). Data dependency graphs for Ada programs. IEEE Transactions on Software Engineering, SE-16(5), 498–509.

Olsen, D. R. (1987). Larger issues in user interface management. Computer Graphics, 21(4), 134–137.

Reingold, E. M., and Tilford, J. S. (1981). Tidier drawings of trees. IEEE Transactions on Software Engineering, SE-7(2), 223–228.

Robins, G. (1987). The ISI Grapher: A Portable Tool for Displaying Graphs Pictorially. Marina del Rey, CA: USC/Information Sciences Institute.

Smith, D. R. (1988, August). KIDS - a knowledge-based software development system. In Lowry, M. (Ed.) Proceedings of the Workshop on Automating Software Design. (St. Paul, MN). (pp. 182—189). Menlo Park, CA: AAAI.

Tamassia, R., Batini, C., and Talamo, M. (1983). An algorithm for automatic layout of entity-relationship diagrams. In C. G. Davis et. al. (Eds.), Entity-Relationship Approach to Software Engineering. Amsterdam: North-Holland, 421–439.

Tichy, W. F., and Newbery, F. J. (1987, September). Knowledge-based editors for directed graphs. In H. Nichols and D. Simpson (Eds.), Proceeding of the 1st European Software Engineering Conference. (Strasbourg, France). (pp. 101–109). Berlin: Springer-Verlag.

Tufte, E. R. (1983). The Visual Display of Quantitative Information. Cheshire, CT: Graphics Press.

Voigt, K., and Tong, C. (1989). Automating the Construction of Patchers that Satisfy Global Constraints. (Working Paper No. 119). Rutgers: Rutgers University, Computer Science Department.

Wetherell, C., and Shannon, A. (1979). Tidy drawings of trees.
IEEE Transactions on Software Engineering, SE-5, 514–520.